Université de Liège

INFO-2051

MOBILE KOMBAT

BAGUETTE Brice(s181482)
GROSJEAN François (s160883)
WERY Victor (s192378)

Master in Civil Engineer

Professor: Laurent Mathy

Academic year 2022-2023

1 Description of the app and rooting

MOBILE KOMBAT is a versus fighting game with an online mode where we can unlock characters and cosmetics with golds earned while playing the game. Cosmetics can be equipped and will modify character stats. The app has a shop, an inventory, a statistic part, and 3 game modes: a standing still bot, a fighting heuristic, and an online mode. The root tree is shown at figure 1

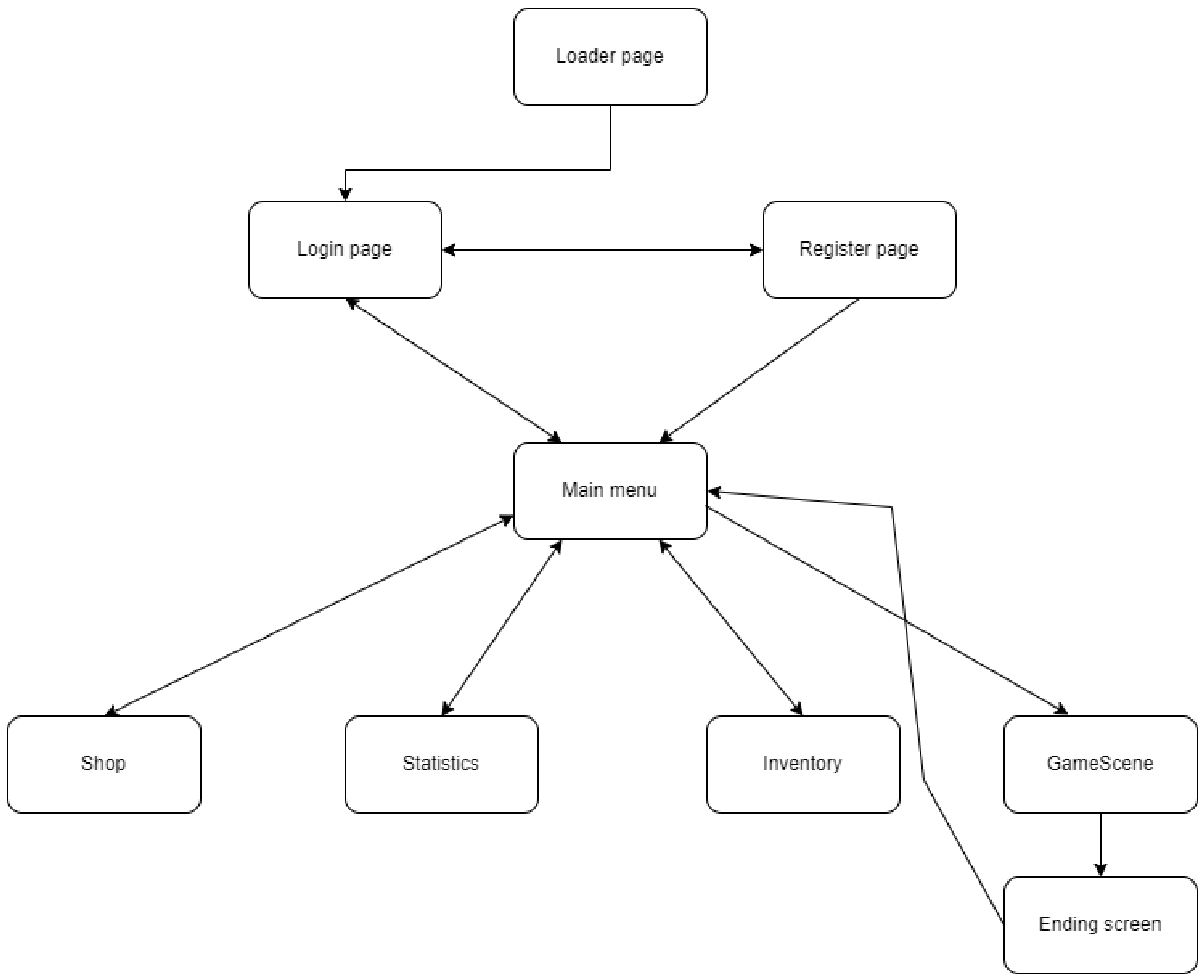

Figure 1: Rooting tree of application

2 Code structure, OO, UML, patterns

The code is divided in 3 parts: controllers, views, and models.
We have 2 controllers in our project, one for the shop and for the inventory, and one for the game management. The first one is used to keep track of user's actions in the inventory and the shop. In the shop, when the user buy an item, the controller will add the item in the inventory, increase the number of items in the statistics of the user, and remove the item from the shop. With the help of the first controller, when an item is equipped, the user can see the look of his character with it and this is also displayed in the main menu. The second one is used to update the game view when the user trigger the different buttons. These interactions are shown in figure 2.

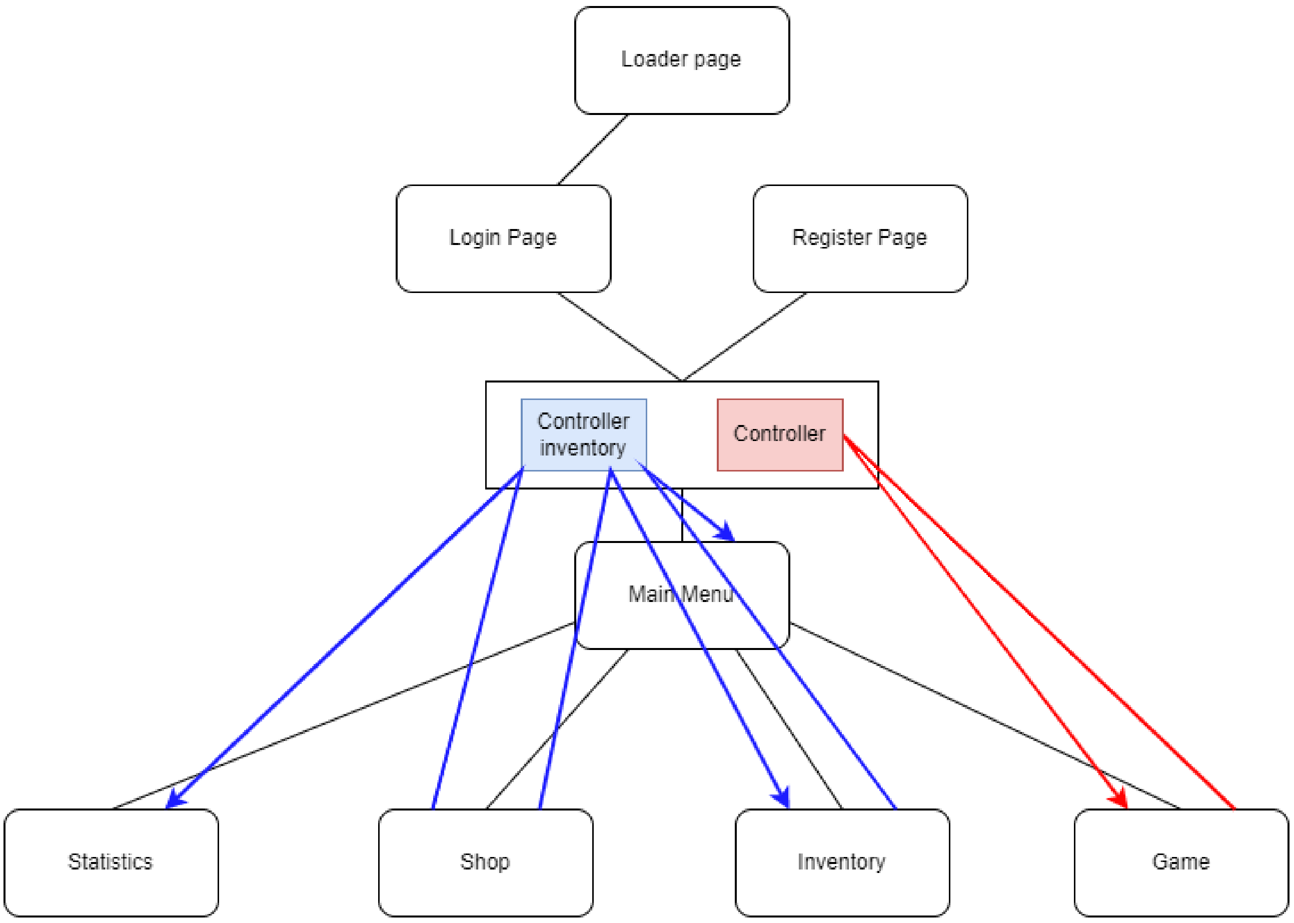

Figure 2: Interactions between controllers and the different views

The models contains all the logic of our applications. The OO schema described below shows it in details.
We also have multiple abstract classes such as 'Button' or 'Opponent' which allows us to define different comportements for the same kind of object. For example in the Opponent class, different class that inherits from it have different behavior for the 'getAction' function. The 'SmartBot' will track the player to try to kill him as fast as he can, the 'dummyBot' won't do anything and the 'RealPlayer' will have info based on realtime database used in the game. Furthermore we had to create 4 classes to handle the online game and encode/decode json to communicate with the database. This is represented in figure 3
We also have multiple singletons such as the loader to get every ui.image used for display in the game or the Constant class to store every constant we have in the game.

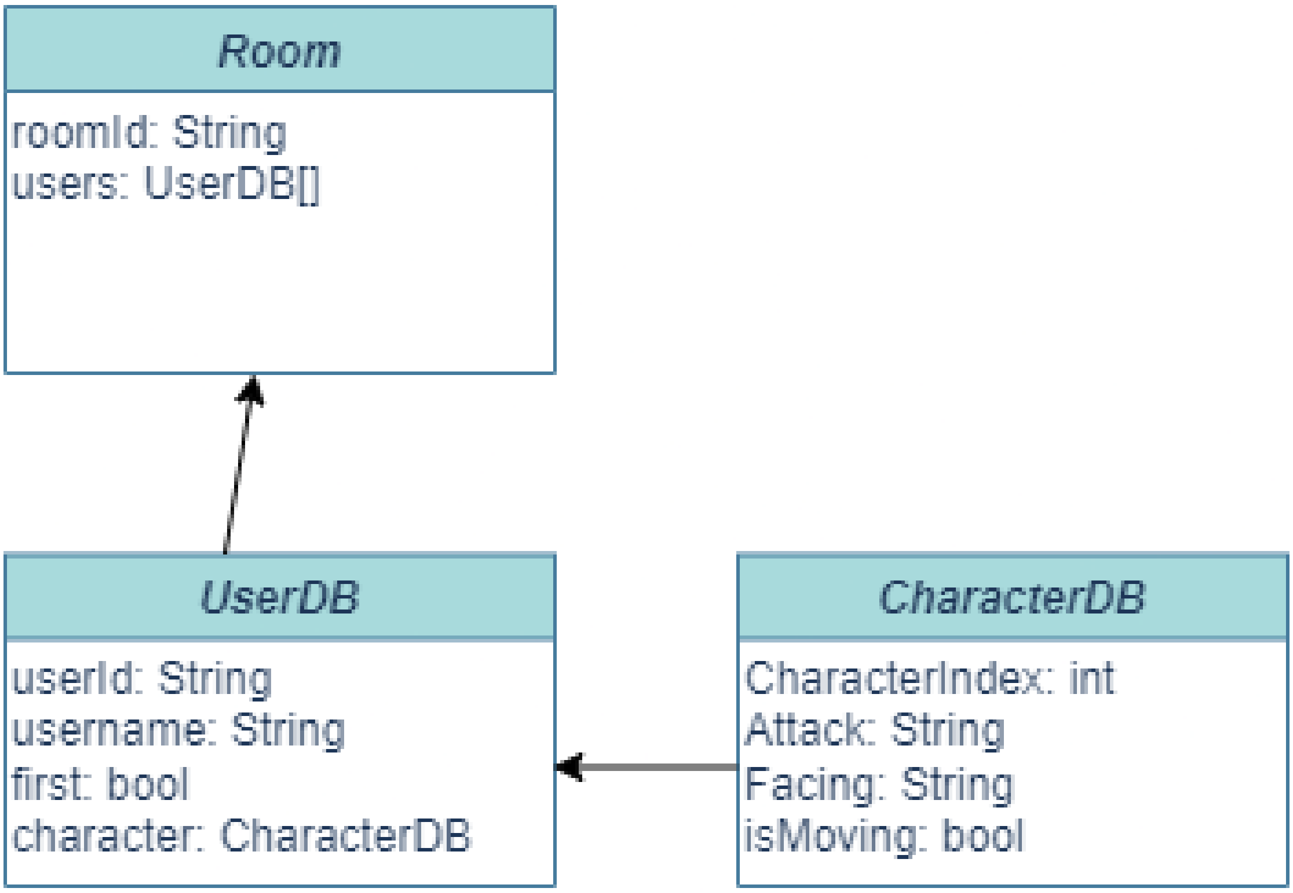

Figure 3: Classes used to handle realtime database data

The figure 4 show the class used for the game management and 5 is more about inventory and shop management.

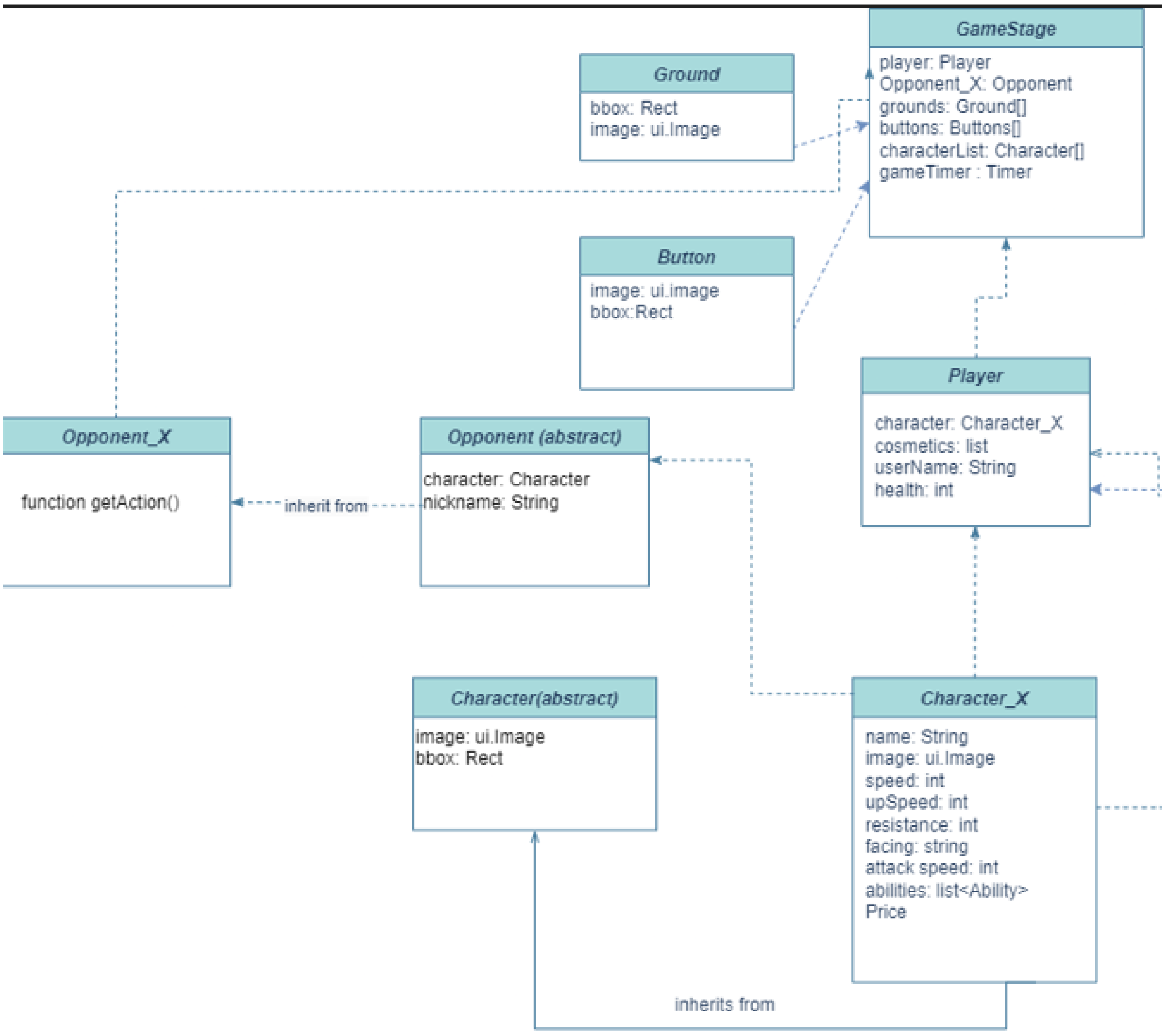

Figure 4: Classes used to handle the game state

For views now, its all about what is gonna be displayed on the screen during the use of application and manage the routing between the different screens.
Starting from the top, the users navigate to the Register/Login page where they can create a new account or connect to theirs and then the users will go to main menu.
In the Main Menu, a user can see his equipped character and cosmetics, and go to Statistics, Inventory, Shop, or can choose to play between 3 game modes that will make him navigate to the GameScene view.
These game modes will be described below. After a game is finished the player will be sent to the EndingScreen where the winner of the fight is highlighted and where the player gets rewards depending on the result of the match.
The Statistics view display some information like the total number of gold earned or the number of characters bought by the user.
In the Inventory, they can change characters and equip them with cosmetics using equip button or by a dragging the cosmetics on top of the appropriate zone.
Finally in the Shop, the users will be able to purchase new characters and cosmetics. We have some customs widgets as well for custom character display etc..

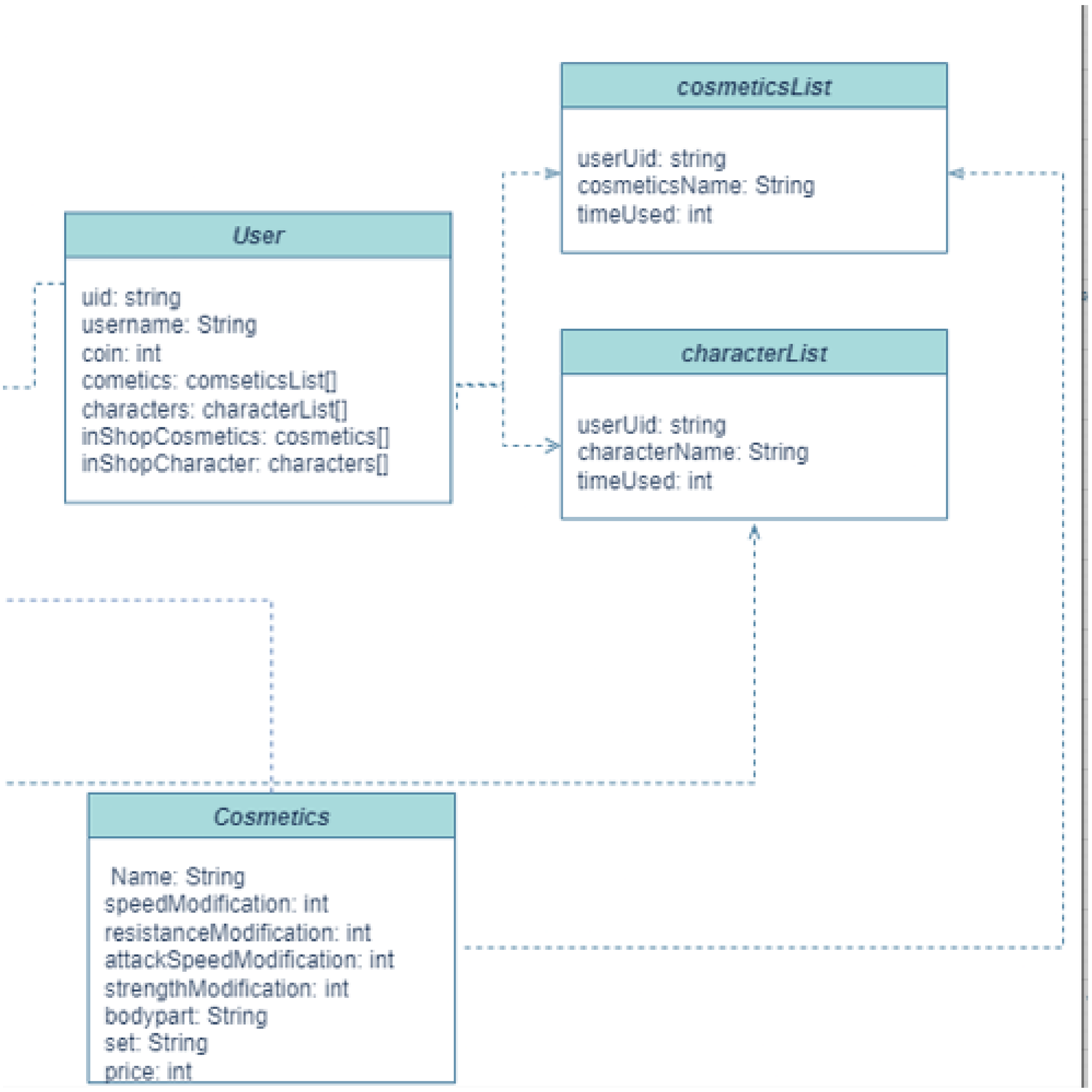

Figure 5: Classes used to handle shop and inventory

2.1 The game itself

The game rules are as such:

Each player is able to move in both direction. He is able to jump as long as he has touched the ground or dodged since his last jump.
Each player is able to attack by typing on the quick attack button at any moment or on the heavy attack button. The latter triggers a different attack depending on if he is jumping, moving, static or floored (typing on the floor button). A character has to finish his ability in progress or get hit to be able to begin a new action.
He is also able to dodge, remaining unaffected by any attack from which he should normally get damages. The dodge ability has a cool down time before which a character is unable to perform a new dodge action.
When a character gets hit by his opponent's attack, he gets some damage depending on the attack as well as a recoil on a certain length depending on the attack and during which he is unable to perform a new action. A character can only get hit once from a same attack iteration from his opponent.

When a character's health falls to 0 or when he falls into the void, the corresponding player has lost the fight and his opponent wins the game and some coins.

The game implements two characters including a human controlled one and one that can be controlled either by a human or a bot. Both characters states are continuously updated either by the buttons for the human or by the heuristics for the bot. Those states determine whether the character is attacking, moving, jumping, etc.

On the display point of view, the game is regularly updated. Characters' position, hit box, image, state are updated given their state just before the display update.
For example, if the character is moving in certain direction, his hit box and image are shifted by a certain amount in that direction, if he was in the air and just touched the ground, he passes from not grounded to grounded.
It is this automatic state update and its rules that defines the limits of a player's liberty to change his character's state. Thus, for example, a player cannot make his character dodge or jump while already using an attack ability. On the other hand, a character is allowed to move while dodging.

3 What does not work and why

We got a timing problem for the submission and when we merged the 2 last parts that were working on their own, i.e. the gameplay and the shop/inventory management system, we got a merge problem and we had to do it quickly 'by hand'. It results in an app that is not crashing but doesn't work as it should. It's a very quick fix that will be done for the presentation but we were out of time at that moment.

On a more technical part, we have some problems of lags with the online and position of players. The goal is to fix this as well for the presentation to get everything working as it should but the offline is working really well.

4 What's missing, problems, technical challenges, etc

On the visual part, many images are still to be included and adjusted for the different animations. The display system of the cosmetics is not working properly on the final version of our code.

On the gameplay point of view, the characters' hit box are still to be adjusted more precisely to their different images and the possibility of falling through the air platform by hitting the floor button should be implemented for the presentation.

The first challenge we faced was to make a proper game loop for a video game which found a solution pretty easily with a Timer.periodic.
The next thing we had as problems and as technical challenge was to understand how to handle online without a proper back-end. Thanks to Firebase we can handle it but was still pretty rough to understand and online is still not perfect.
Another technical problem was to handle the multitude of inputs the applications get like movement and attacks based on it. For the moment the multiple direction for attacks are not well handled.

We also hadn't the time to implement proper actions for each character using the power of abstract class.
Finally the stats part isn't well implemented as well and miss some relations with back-end to work properly, for example to compute the time spent on the app.